RICARDO LUIS DOS SANTOS PROENÇA

LEAN SOFTWARE DEVELOPMENT: CONCEITOS E APLICAÇÕES EM UM ESTUDO DE CASO

Monografia apresentada ao PECE – Programa de Educação Continuada em Engenharia da Escola Politécnica da Universidade de São Paulo como parte dos requisitos para conclusão do curso de MBA em Tecnologia de *Software*.

São Paulo
2013

RICARDO LUIS DOS SANTOS PROENÇA

LEAN SOFTWARE DEVELOPMENT: CONCEITOS E APLICAÇÕES EM UM ESTUDO DE CASO

Monografia apresentada ao PECE – Programa de Educação Continuada em Engenharia da Escola Politécnica da Universidade de São Paulo como parte dos requisitos para a conclusão do curso de MBA em Tecnologia de *Software*.

Área de Concentração: Tecnologia de *Software*

Orientadora: Profa. Dra. Solange Nice Alves de Souza

São Paulo
2013

DEDICATÓRIA

Dedico este trabalho a minha mãe que de qualquer lugar desse universo, está a me olhar.

AGRADECIMENTOS

Ao PECE – Programa de Educação Continuada em Engenharia que ofereceu excelente corpo docente para realização do curso, em especial à professora Dra. Solange Nice Alves de Souza que foi muito importante para que esse trabalho pudesse ser concluído, sem sua paciência, suas orientações claras e precisas, sua disponiblidade para os momentos de dúvida, esse projeto não se materializaria. Também agradeço todo o pessoal administrativo do PECE.

Ao Sandro Prinheiro, proprietário da BeeIT, que aceitou a idéia desse projeto no primeiro momento e sempre esteve a disposição para esclarecimentos necessários.

A minha esposa, Priscila Medeiros Proença, que acordou nas madrugadas para me apanhar na estrada, e por cada dia de malas feitas para que eu pudesse realizar esse curso.

Ao meu pai e minha irmã por fazerem companhia à minha esposa nesse trajeto.

RESUMO

Este trabalho analisa os principais processos de desenvolvimento de *software* ágeis. Em metodologias de desenvolvimento ágil são identificados a filosofia e os princípios que norteiam o processo de desenvolvimento com enfoque em pessoas, comunicação, projetos que priorizam a entrega ante a análise, com enfoque em simplicidade. São identificadas as principais abordagens ageis ou bem próxima delas como o TDD, XP, ASD, DSDM, Crystal, FDD até o *Lean Software Development .* São analisados o contexto de metodologias ágeis e onde o *Lean Software Development* está inserido nesse contexto. Também é apresentado um estudo de caso para identificar se uma empresa que se auto denomina praticante do desenvolvimento ágil pratica o *Lean Software Development.*

ABSTRACT

This work analyzes the main of agile software development processes. In agile development methodologies are identified philosophy and principles that guide the development process with a focus on people, communication, projects that prioritize the delivery before the analysis, with a focus on simplicity. Identified are the main approaches or agile as well as the next one TDD, XP, ASD, DSDM, Crystal, FDD to the Lean Software Development. A study of the context of Agile and Lean Software Development which is inserted in this context. It also presents a case study to identify whether a company that calls itself a practitioner of agile development practices Lean Software Development.

LISTA DE ILUSTRAÇÕES

Pág.

LISTA DE TABELAS

Pág.

LISTA DE ABREVIATURAS E SIGLAS

TI	Tecnologia da Informação
WEB	World Wide Web - rede mundial de computadores
PMBoK	Project Management Body of Knowledge
PMI	Project Management Institute
SLA	Service Level Agreement
CMMI	Capability Maturity Model Integration
UML	Unified Modeling Language
PSP	People Software Process
TSP	Team Software Process
XP	Extreme Programming
ASD	Adaptative Software Development
DSDM	Dynamic System Development Method
FDD	Feature Drive Development
TDD	Test Driven Development
AM	Agile Modeling
BPMN	Business Process Modeling Notation
MO	Medicina Ocupacional
PCMSO	Programa de Controle Médico de Saúde Ocupacional
TFS	Team Foundation Server
ALM	Application Lifecycle Management
EF	Especificação Funcional

SUMÁRIO

1. INTRODUÇÃO

1.1. Motivações

Para entender os métodos ágeis é preciso contextualizar os processos de desenvolvimento ao longo do tempo. Esses processos surgiram justamente como uma reação aos métodos convencionais; sequenciais, representado pelo modelo cascata de desenvolvimento, que mostraram-se ineficientes ou pouco adequados às necessidades atuais.

No final dos anos 90 surgem os métodos ágeis como um agrupamento das antigas e novas práticas em engenharia de *software*, cujo objetivo é a estreita cooperação entre o desenvolvedor e o especialista em negócios, entregas frequentes de valor ao cliente, uma equipe auto-organizada e maneiras padronizadas de criar código, assim como alterações de requisitos com menor impacto no projeto (Agile Alliance, 2012).

Dentro de métodos ágeis existem diferentes abordagens, umas mais próximas do que preconiza o manifesto ágil, outras, com princípios e práticas próprios. O *Lean* é anterior ao manifesto ágil, dessa forma, o *Lean* tenha pode ter sido utilizado de base para a declaração do manifesto ágil.

A prática do Scrum, por exemplo, não assegura que a organização fornecedora de soluções em TI é ágil, a medida que o Scrum não tem o rigor do controle estatístico do processo de desenvolvimento, rigor encontrado no *Lean Software Development*, suas *features*/estórias são tão subjetivas quanto manipuláveis (Middleton & Joyce, 2012). Segue-se o preceito de que o que não é medido não pode ser gerenciado; consequentemente melhorado.

A principal unidade de medida para declarar uma organização ágil é o tempo: tempo entre a solicitação do cliente e a entrega do produto dessa solicitação, para isso o *Lean Sofware Development* tem o *lead time* que é medido a cada iteração, de forma a promover a melhoría contínua, o Scrum tem os backlogs. Destaca-se também a enfase na redução de custos. Dessa forma nota-se mais concistência na adoção da primeira abordagem, pois é capaz de cobrir satisfatóriamente todo o processo de desenvolvimento. Para organizações fornecedoras de soluções em tecnologia pode ser penoso encontrar quais são as funcionalidades que agregam valor ao cliente em um determinado momento, isso considerando o *time to market*. Estudos apontam

que, em alguns casos, somente os métodos ágeis são incapazes de gerar projetos de sucesso (Wang, 2011). Contudo, geralmente, a equipe tende a simplificar localmente o processo de desenvolvimento, assumindo um processo paralelo, gerando uma otimização de parte do processo e não do todo, isso obviamente acaba por descaracterizar o processo, afetando a qualidade do produto. Neste contexto o *Lean Software Development* pode ser empregado como uma abordagem completa, com um conjunto de princípios e práticas que envolvem além de variáveis do processo de desenvolvimento (práticas) como variáveis de como a organização é administrada (princípios), há ainda uma variável que transcende os limites da organização que é o enfoque no cliente, precisa ser do conhecimento de todos na equipe qual é a real necessidade do cliente e o que agrega valor para ele.

A definição de *Lean Manufacturing* para segmentos industriais está muito clara, no entanto, para indústria de *software* as mudanças começaram a partir do título para *Lean Software Development* e ainda hoje não há consenso. Há na literatura uma definição segundo a qual *Lean* é uma filosofia ou um conjunto de princípios que norteiam a prática, e que estes princípios são imutáveis no tempo ou espaço, enquanto que a prática pode ter diferentes aplicações de acordo com o ambiente ou uma situação particular de cada organização (Poppendieck & Poppendieck, 2011). Há também a definição, segundo a qual, *Lean* é um método estatístico rigoroso do processo, com enfoque nas reduções de tempo de ciclo, taxa de erro e variabilidade, enquanto promove a melhoria contínua (Wang, 2011). Neste trabalho o *Lean* é um conjunto de princípios que norteiam a prática que um método estatístico rigoroso do processo.

Na *Agile Conference* realizada, em 2011, *Salt Lake City*, E.U.A o *Lean Software Development* foi considerado como uma área emergente no desenvolvimento de *software*, mesmo sem essa sólida conceitualização acerca da definição do *Lean Software Development*, há evidências da aplicação de *Lean Software Development* puro ou combinado a outras metodologias ágeis com diferentes propósitos (Wang, 2011). Em avaliações de processos empregados em organizações, Wang declara que em algumas é possível identificar o que é *Lean Software Development* e o que é ágil, em outras, os princípios *Lean Software Development* guiam o desenvolvimento e adaptação ao ágil.

A motivação deste trabalho é elucidar o conceito de *Lean Software Development* e o contexto da aplicação em uma organização, tendo como *benchmarking* estudo similares, como o de Middleton & Joyce (2012) (2012), no qual são apresentados excelentes resultados no emprego dessa abordagem.

1.2. Objetivo

O objetivo deste trabalho é avaliar o emprego do *Lean Software Development*, bem como a combinação com métodos ágeis, no processo de desenvolvimento de *software* em uma organização fornecedora de serviços de tecnologia.

Será realizado uma experimentação numa organização fornecedora de soluções em tecnologia, cujo *core-business* é voltado para soluções *web*. Neste estudo será avaliado a aderência da organização aos sete princípios e práticas *Lean Software Development.*

A proposta é solidificar o conhecimento em *Lean Software Development.* Discutir a aplicação da abordagem, sua filosofia, seus principios e práticas no desenvolvimento de *software*, com princípios clássicos e validados em outros segmentos industriais, de forma que os desenvolvedores gostem da abordagem e ao mesmo tempo gerem valor para o cliente e remunere os acionistas (Middleton & Joyce, 2012).

Após a realização desse trabalho espera-se que a organização tenha uma definição do conceito de *Lean Software Development* e um diagnóstico do quão aderente está em relação aos seus princípios e práticas, podendo optar por dar sequência à aborgagem, ou pela sua combinação com outras metodologias ágeis de desenvolvimento de *software*.

1.3. Justificativas

De acordo com o PMBoK (*Project Management Body of Knowlege)*, projeto é todo esforço necessário para criar um produto (*software*) ou resultado exclusivo (PMI, 2004). Um projeto de *software* não é exatamente igual a outro, mas apresenta características similares, sendo então possível identificar fluxos de processos que atendam a um agrupamento, considerando as características e necessidades do produto de *software* e da equipe.

Em um processo de desenvolvimento de *software*, toda equipe encontra dificuldade a medida que avança no desenvolvimento (Pressman, 2009). Assim, a divulgação de soluções, com o relato de problemas, riscos, caminhos e resultados seria de grande utilidade, pois guiariam equipes de desenvolvimento na localização e resolução rápida dos problemas.

Supõe-se que a organização adepta de algum processo de desenvolvimento ágil em alguma medida pratica a abordagem de desenvolvimento *Lean Software Development.* No entanto, não apresenta o mesmo enfoque no controle estatístico do processo e em melhoria contínua. Como afirma (Wang, 2011), as aboradgens podem não estar bem distinguidas para a organização.

Um trabalho similar realizado em Londres evidenciou que a implementação do *Lean Software Development,* após o período de doze meses, melhorou o tempo de entrega em 37%, a consistência das entregas subiram 47% e os defeitos relatados pelos clientes caíram 24% ((Middleton & Joyce, 2012), 2012).

Para a conjuntura econômica brasileira e mundial, o nível de exigência dos usuários, os atributos de qualidade esperados e percebidos e a concorrência global fazem com que as equipes de desenvolvimento de *software* tenham que atender às mudanças impostas pelos *stakeholders* e entregar de forma ágil, dentro do tempo acordado o que lhe foi solicitado.

O *Lean Software Development* se coloca como uma importante aborgagem pela importância das equipes que se auto-organizam, a comunicação com o cliente, assim como a colaboração e ênfase na entrega rápida de valor para o cliente, além de redução de custos.

A ênfase está na melhoria contínua do fluxo de valor para a organização fornecedora de serviços ou do departamento de Tecnologia da Informação. Os componentes são mensurados pelo tempo de ciclo e não pelas estórias como em outras metodologias ágeis.

A velocidade na entrega é determinante para o sucesso do *Lean Software Development.* A ausência de desperdício proporciona velocidade e o tempo de ciclo é um alerta quando algo está errado. A organização fornecedora de serviços em

tecnologia precisa transformar as necessidades identificadas do cliente em valor entregue a ele em uma cadência confiável e repetitiva. Diante disso os gerentes tem condição de discutir e propor o *SLA - Service Level Agreement* mais efetivo.

Lean Software Development poder ser implantado com diferentes propósitos, (Wang, 2011):

- No sentido *top-down* para que toda a organização conheça de modo mais claro o ciclo de desenvolvimento do departamento de Tecnologia da Iinformação;
- Como facilitador na comunicação com diferentes *stakeholders* – a medida que o cliente enxerga valor de forma mais rápida;
- A prática *Lean Software Development* no sentido mais prescritivo pode auxiliar na melhoria contínua do processo e ainda contribuir para certificações CMMI, para a organização que almejam o nível quatro, por exemplo.

De modo geral podemos ter o *Lean Software Development* implementado como abordagem secundária dentro de qualquer outra metodologia de desenvolvimento ágil ou o contrário, qualquer outra metodologia de desenvolvimento dentro *Lean Software Development*.

1.4. Metodologia da pesquisa

Além da pesquisa de referências bibliográficas, houve também a pesquisa e coleta de dados, assim, o trabalho dividiu-se em duas etapas:

A primeira foi a coleta de dados referentes aos chamados no OComom com status de “fechado” no período de 13/08/2010 a 14/08/2012.

A segunda foi a realização de questionários aplicados à empresa. Após a compilação da informação, o material foi analisado e o texto começou a ser escrito. Periodicamente uma versão foi submetida à orientadora para avaliação, correção, sugestão, a cada interação, ajustes foram feitos no texto até a versão final.

Incialmente estava previsto a aplicação do questionário a todos os funcioários da BeeIT, o que não foi possível, apenas o proprietário da empresa foi entrevistado.

Uma visita a empresa para observação *in loco* também estava prevista, mas também não ocorreu.

Destaca-se que no momento da elaboração desta monografia, o autor deste trabalho era funcionário da empresa cliente, e sob essa ótica a BeeIT foi analisada.

1.5. Estrutura do Trabalho

O Capítulo 1 **INTRODUÇÃO** apresenta as motivações, o objetivo, as justificativas, a metodologia da pesquisa e a estrutura do trabalho.

O Capítulo 2 **INTRODUÇÃO AOS PROCESSOS DE DESENVOLVIMENTO** apresenta os principais modelos de desenvolvimento de *software* ágil ao longo do tempo com enfase em *Lean Software Development.*

O Capítulo 3 ***LEAN SOFTWARE DEVELOPMENT*** apresenta a origem do *Lean, suas variações como Lean thinking, Lean Manufaturing, Lean Startup e Lean Software Development,* este último é o enfoque da exploração conceitual, seus diferentes tipos de aplicação.

O Capítulo 4 **ANÁLISE DA ORGANIZAÇÃO SOB A ÓTICA DOS PRINCÍPIOS *LEAN SOFTWARE DEVELOPMENT*** descreve uma organzação fornececedora de serviços em tecnologia, a BeeIT, e a caracteriza segundo os sete princípios e práticas *Lean Software Development.*

O Capítulo 5 **CONSIDERAÇÕES FINAIS** descreve a conclusão desse trabalho, sugestões para trabalho.

REFERÊNCIAS BIBLIOGRÁFICAS relaciona os artigos, livros, teses e dissertações utilizados nesse trabalho para fundamentação teórica.

APÊNDICE A indaga sobre os processos da Beeit com com cliente para levantamento do cenário.

APÊNDICE B esmiuça a relação contratual entre a Beeit e o cliente.

APÊNDICE C relaciona os princípios e práticas *Lean Software Development* com a empresa analisada.

2. INTRODUÇÃO AOS MODELOS DE PROCESSO DE DESENVOLVIMENTO

Nos últimos cinquenta anos os processos de desenvolvimento de *software* evoluíram concomitantemente às exigências do mercado, dos clientes e da complexidade das aplicações desenvolvidas. O processo de desenvolvimento de *software* define as atividades, ações e tarefas necessárias para desenvolver um *software* de qualidade, (Pressman, 2009). Atividades de processos de *software* como comunicação, planejamento, modelagem, construção e entrega estão presentes em qualquer projeto de *software*.

Tão importante quanto o processo de desenvolvimento é o fluxo desse processo em relação à sequência das atividades e ao tempo de execução. Sendo assim, pode-se ter um processo linear, iterativo, evolucionário ou processo paralelo. Neste capítulo aborda-se os processos de desenvolvimento com enfoque aos processos ágeis.

2.1. Processo de desenvolvimento ágil

O desenvolvimento ágil, surgido em 2001 com a publicação do manifesto ágil (Poppendieck & Poppendieck, 2011), combina uma filosofia com um conjunto de princípios de desenvolvimento. A **filosofia** defende a satisfação do cliente e a entrega incremental de partes do produto, para isso, defendem como parte da filosofia equipes de projeto pequenas e altamente motivadas; métodos informais; artefato de engenharia de *software* mínimo e, acima de tudo, simplicidade no desenvolvimento. Os **princípios** de desenvolvimento priorizam a entrega mais que a análise e projeto, o que não significa que essas atividades devam ser eliminadas, mas sim adequadas de forma que a entrega seja realizada dentro do prazo estabelecido. A comunicação ativa e contínua entre desenvolvedores e clientes é fundamental no processo.

No modelo de desenvolvimento ágil prima-se pela capacidade de resposta às mudanças quando elas são necessárias e adaptação incremental, utilizando-se do *feedback* de clientes para que a melhor solução seja empregada. As respostas as mudanças são economicamente mais viáveis em relação aos modelos de processos de desenvolvimento de *software* tradicionais (Agile Alliance, 2012).

A *Agile Alliance,* organização que explora e aplica os princípios e práticas ágeis com intuito de fazer a indústria mais produtiva, humana e sustentável, enumera 12 princípios para a agilidade, os quais são apresentados a seguir. Eles não são completamente seguidos em todos os processos de desenvolvimento ágil, mas a relevância do item para a organização determina o que ela está buscando (Agile Alliance, 2012).

1. A maior prioridade deve ser a satisfação do cliente por meio de entrega adiantada e contínua de partes do *software* que sejam úteis para o cliente.
2. Os pedidos de alterações devem ser bem acolhidos, mesmo que o desenvolvimento esteja atrasado. Os processos ágeis se aproveitam das mudanças como uma vantagem competitiva na relação com o cliente.
3. *Software* deve ser entregue frequentemente em intervalos de tempo mais curtos.
4. O pessoal comercial e os desenvolvedores devem trabalhar ao longo de todo o projeto diariamente em conjunto.
5. Desenvolvedores devem estar motivados. Ambiente estimulador, apoio e confiança no trabalho dos desenvolvedores são fundamentais no processo.
6. O método mais eficiente e efetivo para transmitir informações para dentro de uma equipe de desenvolvimento é uma conversa aberta e presencial.
7. *Software* em funcionamento é a principal medida de progresso.
8. Os processos ágeis promovem desenvolvimento sustentável. Os proponentes, desenvolvedores e usuários devem estar capacitados para manter um ritmo constante sempre.
9. Atenção contínua à excelência técnica e bons projetos aumenta a agilidade.
10. Simplicidade - a arte de maximizar o volume de trabalho que não precisou ser feito – é essencial.
11. Equipes que se auto-organizam conseguem definir melhores arquiteturas, especificar requisitos mais corretamente e desenvolver projetos de melhor qualidade.
12. Equipes se avaliam em intervalos regulares, sintetizando e ajustando seu comportamento tornam-se mais eficientes.

Destaca-se a importância do fator humano, o processo é que deve ser adaptado à equipe e às pessoas e não o contrário. Deve existir uma sinergia entre as pessoas que formam a equipe, sendo atributos de destaque: a competência, o foco comum, a colaboração, a habilidade na tomada de decisões e solução de problemas confusos, a confiança mútua, o respeito e a auto-organização.

***Extreme Programming* – XP:** emprega um conjunto de regras e práticas constantes no contexto de quatro atividades metodológicas: planejamento, projeto, codificação e testes, além de valores como base do trabalho a ser realizado: comunicação, simplicidade, *feedback*, coragem e respeito. As principais diferencias dessa abordagem são a implementação com o mínimo de documentação e a execução somente daquilo que foi solicitado pelo cliente, sendo desencorajada a inclusão de funcionalidades que são supostas serem necessárias no futuro. Existe uma variação dessa abordagem que é a Industrial XP que é imbuída de espírito minimalista, centrado no cliente e orientado a testes, difere principalmente por seu enfoque em gerenciamento e por suas práticas atualizadas (Pressman, 2009).

Desenvolvimento guiado por testes (*TDD - Test Driven Development*): requisitos de um componente de *software* são a base para a criação de uma série de casos de testes que simulam a interface na tentativa de encontar erros na estrutura de dados e nas funcionalidades entregues por componente. É uma aboragem que sugere ciclos curtos de retição e que os testes sejam escritos antes da codificação, com enfase em baixo acoplamente e alta coesão. Tem aspectos diferentes na abordagem (Beck, 2003)dos métodos ágeis mas não são excludentes (Beck, 2003).

Desenvolvimento de *Software* Adaptativo (ASD - *Adaptative Software Development*)**:** técnica para construção de *softwares* e sistemas complexos, suas bases filosóficas se concentram na colaboração humana e na auto-organização das equipes. Incorpora no seu clico de vida três fases: especulação, colaboração e aprendizagem (Pressman, 2009).

***Scrum**:* os princípios do *Scrum* são consistentes com o manifesto ágil e são usados para orientar as atividades: requisitos, análise, projeto, evolução e entrega. Em cada atividade são especificadas um conjunto de tarefas que seguem o padrão de processo conhecido como *Sprint.* O trabalho realizado dentro de um *Sprint* é

adaptado ao problema em questão, sendo muitas vezes modificado em tempo real, pela equipe *Scrum* – o numero de *sprints* para cada atividade varia de acordo com o tamanho e a complexidade do produto (Pressman, 2009).

Métodos de desenvolvimento de sistemas dinâmicos (*DSDM – Dynamic System Development Method*): metodologia voltada para a construção e manutenção de sistemas que atendam restrições de prazos, através do uso de prototipagem incremental em um ambiente de projeto controlado. A filosofia baseia-se em uma versão modificada do principio de Pareto - 80% da aplicação pode ser entregue em 20% do tempo que levaria para entregar a aplicação completa, cada iteração segue este principio (Pressman, 2009).

Crystal: concebido visando a elaboração de uma abordagem de desenvolvimento de *software* que priorizasse a adaptabilidade durante a iteração, tendo como objetivo a entrega parcial de *software* útil e a preparação para a segunda iteração (Pressman, 2009).

Desenvolvimento guiado à funcionali*dade* *(FDD - Feature Driven Development*): sua filosofia é enfatizar a colaboração entre os integrantes da equipe, gerência de problemas e complexidade dos projetos, utilizando a decomposição baseada em funcionalidades, seguida pela integração dos incrementos e comunicação de detalhes técnicos utilizando-se de meios verbais, gráficos e de texto. Atividades de garantia da qualidade são também enfatizadas, através de inspeções de código e projeto, coleta de métricas e o uso de padrões. O FDD é mais voltado às diretrizes e técnicas de gerenciamento de projetos do que outros métodos ágeis (Pressman, 2009).

Modelagem Ágil (*AM - Agile Modeling*): consiste num conjunto de valores, princípios e práticas voltados à modelagem do *software*. Ressalta-se neste modelo a importância dos especialistas em negócios e demais envolvidos, o profissional de tecnologia não tem todas as respostas, para isso o modelo sugere: modele com um objetivo, use modelos múltiplos, conteúdo é mais importante que a representação, tenha conhecimento, domínio dos modelos e das ferramentas que for utilizar e adapte localmente (Pressman, 2009).

***Lean*:** concebido depois da segunda guerra mundial, na Toyota, por Taiichi Ohno (1912 - 1990), então executivo da empresa, o Sistema Toyota de Produção conhecido como *Lean Manufacturing* ganhou força a partir da década de 90 na indústria automotiva (Poppendieck & Poppendieck, 2011).

A princípio, as evidências do *Lean* despertaram interesses somente de outras companhias automotivas como a Porsche, na Alemanha. Mas quando a Toyota lançou a primeira fábrica no ocidente e todos esperavam que ela utilizasse em seu processo de produção técnicas ocidentais, um novo paradigma começou a ser delineado para a produção de produtos ((Middleton & Joyce, 2012), 2012). A Figura 1 apresenta que na família *Lean* existe uma abordagem para produção enxuta, voltada para operações e outra para desenvolvimento de produto na qual se insere o desenvolvimento de software.

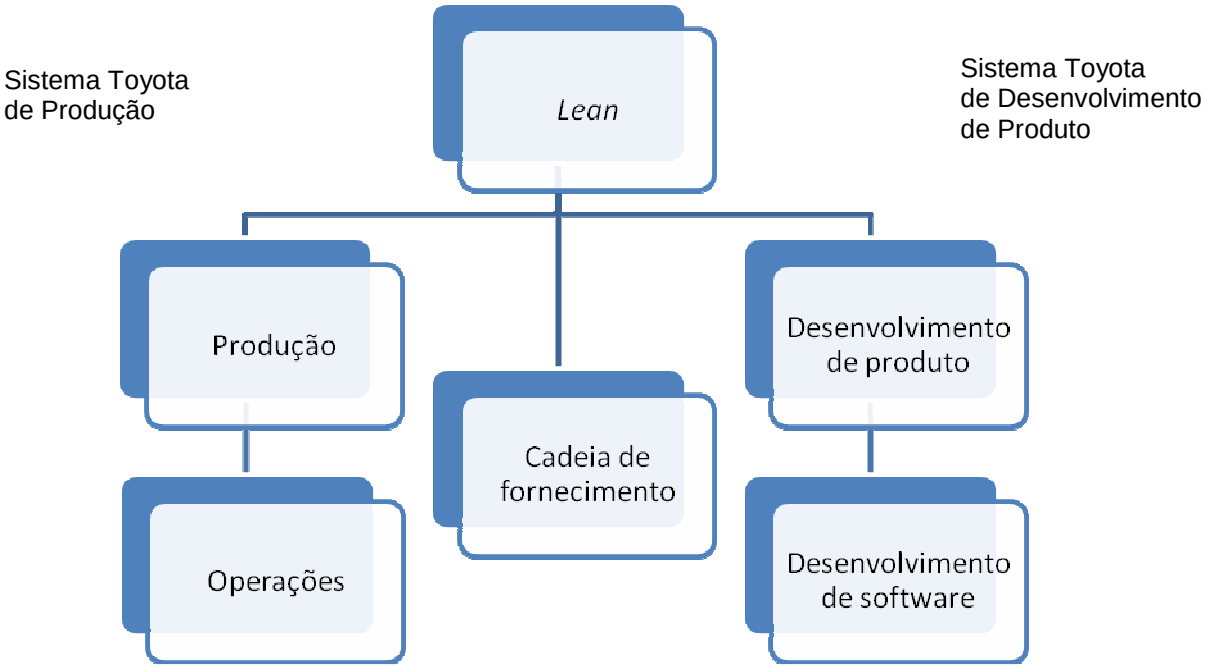

Figura 1 – A árvore da família *Lean* - (Poppendieck & Poppendieck, 2011)

A partir deste momento a prática Lean foi disseminada para outros seguimentos industriais como o aeroespacial (Boeing, EUA). Hoje há evidencias da aplicação dos princípios *Lean* nos mais diferentes segmentos, incluindo desenvolvimento de software, abordado em mais detalhes no próximo capítulo.

2.2. Considerações do capítulo

Constata-se a evolução dos processos de desenvolvimento de software ao longo do tempo concomitantemente às exigencias do mercado, dos clientes e da complexidade das aplicações.

As etapas básicas do processo de desenvolvimento de software (comunicação, planejamento, modelagem, construção e entrega) estão presentes em qualquer abordagem de desenvolvimento, ora com ênfase em uma etapa ora em outras (Pressman, 2009).

A análise dos principais processos de desenvovimento neste estudo permite notar a simbiose entre os processos de desenvolvimento, eles surgem, comumente, como uma evolução da abordagem anterior e não são totalmente antagônicos. A quebra de paradigma ocorre com o surgimento dos métodos ágeis de desenvolvimento. No entanto, dentro dos métodos ágeis essa mesma simbiose é percebida entre as diferentes abordagens de metologias ágeis.

O *Lean* surge na década de 60 na Toyota, depois se espalha para fora do Japão e outros segmentos industriais com sucesso. Na década de 90 surge o desenvolvimento ágil com a publicação do manifesto ágil, com muitos dos atributos do *Lean*, como por exemplo, entrega rápida, equipe comercial e engenharia trabalhando próximos, equipe motivada, a mudança de processo de um produto para outro é uma vatagem competitiva, método eficiente de transmitir informação e fazer o essencial para o cliente. O *Lean* também se caracteriza pela preocupação com fator humano e a avaliação do contexto da organização (Poppendieck & Poppendieck, 2011).

Para este trabalho, o desenvolvimento ágil de software e seu manifesto ágil surgiram da contibuição do *Lean Manufacturing* na industria automoblistica para o desenvolvimento de software, inclusive, esse comportamento é identificado em outros segmentos industrais (Katayama, 2010).

3. LEAN SOFTWARE DEVELOPMENT

O *Lean Software Developement* surgiu como mais uma opção entre os já conhecidos processos de desenvolvimento de *software*. Por ser menos prescritivo seus princípios têm sido associados a outras metodologias de desenvolvimento ágil. Apesar de ser uma abordagem capaz de cobrir todo o ciclo de desenvolvimento de *software*, desde a concepção do projeto até o uso do sistema em produção, o processo é vagamente descrito quanto a sua implementação (Katayama, 2010).

A abordagem *Lean Software Developement* considera todo o contexto da organização e não apenas aspectos técnicos. A principal vantagem e limitação à implementação desta abordagem é que não há imposição à adoção de ferramentas, técnicas ou processos, por esse motivo, não existe um roteiro pronto em como aplicar esses conceitos. Como afirma ((Middleton & Joyce, 2012), 2012) o *Lean Software Development* não substitui práticas tradicionais de engenharia de *software*. Mas de acordo com os artigos (Swaminathan & Jain, 2012), a combinação do *Lean Software Development* com as melhores práticas de engenharia trazem benefícios para o fluxo. (David Raffo, 2010) relata que a combinação do *Lean Software Development* além promover melhorias no fluxo de trabalho, também facilita o gerenciamento tático do projeto.

Os princípios e as práticas do *Lean* são detalhados a seguir. Para a avaliação da organização quanto a sua aderência ao *Lean Software Development*, é fundamental o entendimento desses princípios.

Princípios são verdades subjacentes que não mudam com o tempo ou espaço, enquanto as práticas são aplicações dos princípios a uma situação particular. As **práticas** podem e devem diferir conforme se muda de um ambiente para outro, e elas também mudam a medida que a situação evolui (Poppendieck & Poppendieck, 2011).

Os princípios ajudam a ter uma visão mais concreta dos valores, além de fornecer diretrizes para tomadas de decisões. Por outro lado, as práticas oferecem uma

direção específica sobre o que fazer, mas necessitam de adptação de acordo com o domínio.

3.1. Os sete princípios do desenvolvimento *Lean Software Development* e suas práticas

3.1.1. Eliminar o desperdício

É a essência do *Lean*, do momento da requisição do cliente até a entrega de uma solução. O que não agrega valor ao cliente é desperdício e deve ser eliminado ao longo da linha do tempo.

No entanto para esse processo apresentar resultados efetivos é necessário estar muito claro qual é fluxo de valor. Na relação cliente-fornecedor, fluxo de valor são todos os eventos na linha do tempo que geram vantagem ao cliente, ou seja, que impulsione seu negócio.

Dada uma solicitação de serviço feita pelo cliente, **fluxo de valor** compreende o conjunto de fases e atividades necessárias à execução desse serviço e o tempo despendido até o seu atendimento efetivo (solucionado). O conjunto de fases, atividades e o tempo gasto são definidos como mapa de fluxo de valor. Assim, um mapa de fluxo de valor começa no cliente e termina quando uma solução, um resultado ou produto é entregue satisfatoriamente a esse cliente. Tudo o que não agrega valor a este fluxo é desperdício. Conseguir distinguir no processo as etapas que agregam valor das etapas desperdícios é o desafio contínuo que o fornecedor deve fazer sob a perspectiva do cliente.

Shigeo Shingo na Toyota identificou os sete principais tipos de desperdícios que ocorriam em manufatura e posteriormente Poppendieck & Poppendieck (2011) fez um paralelo desses desperdícios em relação ao desenvolvimento de software, os quais são apresentados na Tabela 1.

Tabela 1 - Os sete tipos de desperdícios na manufatura e no desenvolvimento.

Na Manufatura.	**No desenvolvimento de software**
Inventário	Trabalho parcialmente feito
Superprodução	Funcionalidade extra
Processamento extra	Reaprendizado
Movimentação	Handoff
Transporte	Multitarefa
Tempo de espera	Tempo de espera
Defeitos	Defeitos

As principais fontes de desperdício em desenvolvimento de software são:

Trabalho inacabado: todo projeto tem começo, meio e fim, se um artefato começa a ser desenvolvido ele deve ser documentado, testado e entregue o mais breve possível, segundo o planejamento. Para isso ser possível o trabalho deve estar dividido em pequenos lotes ou iterações.

Funcionalidades extras: se não foi solicitada tal funcionalidade ela não deve ser desenvolvida. Todo desenvolvimento deve ter uma necessidade clara de negócio e sua viabilidade econômica comprovada.

O pior dos desperdícios é gerar funcionalidades extras supondo uma necessidade futura. Todo código criado custa seu desenvolvimento atual e sua manutenibilidade futura.

Reaprendizagem: é alto o custo da reaprendizagem, ou seja, retomar aprendizados passados depois de certo tempo. Por outro lado, construir software é gerar conhecimento. Entende-se que o trabalho divido em iterações leva à prática mais constante, ou em tempo menor, do

conhecimento adquirido, diminuindo o tempo para retomar àquele conhecimento e consequentemente, o custo da reaprendizagem.

Transferência de controle: é a tradução literal para *Handoffs*. Muito do conhecimento sobre determinada coisa é tácito e está na cabeça de seu criador, e à medida que se opta pela transferência do trabalho, ao menos 50% do conhecimento é deixado para trás a cada transferência (Poppendieck & Poppendieck, 2011). A documentação por si só é insuficiente para transferir o controle do trabalho a alguém. Assim, devem-se incorporar outras ações, além da documentação, para transferências de conhecimento mais efetivas.

Troca de tarefas: o desenvolvimento de software requer muita concentração e reflexão para lidar com a complexidade inerente ao processo. Assim, quanto menor for a troca de tarefas melhor, pois isso gera distração que pode ocasionar em defeitos. Essa variável está diretamente ligada a lotes menores e ao tempo de reaprendizagem. Faz todo sentido ter uma tarefa completa do que várias em andamento.

Atrasos: Esta variável está fortemente relacionada à transferência de controle. É difícil encontrar em qualquer organização todo o conhecimento documentado, ter um ambiente favorável à troca de informação evita atrasos. Precisa estar claro para os membros da equipe onde encontrar a informação necessária e a postura de parar por falta de informação deve ser desencorajada.

Defeitos: o objetivo é desenvolver software a prova de falhas. Se defeitos na verificação final são apresentados costumeiramente é porque o processo de desenvolvimento é vicioso. Testes unitários e exploratórios devem ser realizados tão cedo quanto possível e se encontrado o erro, ele deve ser documentado para não ocorrer em testes futuros.

3.1.2. Integrar a qualidade

É fazer o certo da primeira vez, é a disciplina para construir qualidade; pois retrabalho não constrói qualidade em um produto de *software*.

As palavras velocidade e qualidade podem para alguns apresentar conceitos diferentes, para o *Lean Software Development* é impossível haver velocidade sem qualidade. Um dos princípios do *Lean*: se um erro grave é encontrado durante a fase de testes deve-se parar o projeto – é um alerta para equipe de que algum desvio houve do planejamento inícial e o projeto precisa ser revisto, a equipe é obrigada a parar até que corrija o código e refaçam os testes.

Para que o princípio da qualidade ocorra é essencial que se tenha disciplina, pois proporcionará velocidade com qualidade. Contudo, no contexto de desenvolvimento *software*, deve-se aliar outros itens como Poppendieck & Poppendieck (2011): programas períodicos de ***housekeeping***, ou seja, deve-se ter uma estrutura que auxílie o trabalho, não só no ambiente físico como salas ou estações de trabalho, mas também em ambientes virtuais como *desktops*, servidores, diretórios de projeto, etc. A equipe deve ser estimulada a pensar, sempre tendo um ambiente mais limpo, simples, lógico e fácil para encontrar algo. Uso de **padrões** para reduzir o tempo de conversão e a complexidade, além de facilitar a comunicação com stakeholders e o entendimento por futuros membros da equipe.

Segundo Poppendieck & Poppendieck (2011), padrões para uma organização *Lean* incluem convenções de nomeclatura, padrões de codificação, convenção de interação com usuário, estruturas de arquivos, práticas de gestão de configuração, ferramentas, padrões para registro de erros e padrões de segurança; evidentemente, exitem outros como, por exemplo, arquitetura, projetos, etc. Os padrões devem ser seguidos sempre e a equipe deve ser encorajada a mudá-los sempre

que uma melhor alternativa for comprovadamente encontrada. Se um novo padrão entra em operação, tem-se melhoria do processo.

3.1.3. Criar conhecimento

É durante a codificação que os problemas serão apresentados e o conhecimento do projeto será alcançado de forma ampla com *feedback* dos clientes, não desperdiçando tempo com a geração de documentação prematura. Contudo, o conhecimento precisa ser registrado e disseminado, mas de forma simples e rápida. Assim, o colaborador é incentivado a resolver problemas por meio de métodos científicos e suas decisões devem ser documentadas preferencialmente utilizando-se de tabelas, figuras, gráficos em uma **folha A3**. Um exemplo de incentivo à criação de conhecimento é o ***hackathon*** que consiste em liberar os programadores após a *release* para projetos pessoais relacionados a área de atuação. Caso algum *bug* seja reportado, para-se obrigatóriamente e faz um *patch* de correção.

Em uma organização *Lean* o processo de melhoría contínua é sistemático e um problema de cada vez é resolvido, em uma cadência certa e esperada. Entende-se que a equipe só criará conhecimento se existir em sua alocação a previsão de horas destinadas ao processo de melhoría contínua. A equipe não será crítica em relação ao processo se sua única preocupação for a entrega a qualquer custo para satisfazer o cliente.

3.1.4. Adiar comprometimentos

Deixar para tomar uma decisão importante, antes que seja tarde, porém quando um maior número de informação estiver disponível (Poppendieck & Poppendieck, 2011).

O planejamento se faz necessário para o processo de desenvolvimento de *software*, contanto que não contemple decisões prematuras, pois é um modo falho de planejamento, pois agrava o imapcto de defeitos,

limita a utilidade do produto e aumenta o custo das mudanças. É inconcebível gerenciar complexidade sem um plano de execução das tarefas a serem realizadas. No entanto, do ponto de vista deste princípio, as iterações e incrementos são valorizados a ponto de que todo o planejamento é sucetível à mudanças com o avançar do projeto. Como afirma (Poppendieck & Poppendieck, 2011), a melhor estratégia é evitar generalizações desnecessárias e fazer com que o sistema seja flexível apenas nas áreas mais propícias à mudança.

O planejamento é realizado de modo a sempre ter uma estratégia em aberto para o contexto sobre o qual não há uma definição muito clara da decisão. Neste sentido o *Lean Software Development* evita a paralisia da 'análise' e define que uma decisão importante e irreversível deve ser tomada não tão cedo e não tão tarde.

Como alternativa ao modelo tradicional de plano de negócios apresenta-se o ***business model canvas***, (Business Model Generation, 2012) que é um **mapa visual** dos principais elementos de uma empresa, que compreendem parceiros e atividades chaves, proposição de valor, relacionamento com cliente, segmento dos clientes, recursos chaves, estrutura de custos e fluxo de receitas que facilita a divergência ou a convergência.

3.1.5. Entregar rápido

Quanto mais rápida for a entrega da solicitação menor será a volatilidade nos requisitos por parte dos clientes.

A velocidade da entrega está diretamente ligada a qualidade e geração de valor para o cliente, sem desperdícios. O ***lead time*** é uma medida universal em *Lean Software Development* e significa o tempo decorido entre a solicitação de um pedido e sua respectiva entrega. A organização madura é aquela que consegue entregar, repetidas vezes, na cadência e com previsibilidade.

Neste sentido ainda tem-se uma vantagem em entregar rápido que é inerente ao *time-to-market*, com as mudanças de requisitos cada vez mais constantes e frequentes, a organizaçao lenta pode estar desenvolvendo uma funcionadade cuja necessidade não se faz oportuna. Ou, quando o cliente não sabe exatamente o que necessita, a entrega rápida poderia auxiliar no processo de amadurecimento da ideia.

Algumas práticas que apoiam este princípio são:

Folga: uma organização *Lean* não trabalha em sua capacidade extrema.

Uma folga no planejamento deve existir para evitar atrasos. Além de prover ócio criativo para as pessoas pensarem em como entregar mais rápido na próxima iteração. Conforme afirma Demarco (2011), possuir folga na organização oferece a capacidade de realizar mudanças, se reinventar e mobilizar recursos para crescer.

Iteração: A iteração é importante pois a cada ciclo completo de *design-code-verify-release* um *feedback* do cliente é colhido. Só a partir desse momento é que novas estórias são planejadas e quebradas em tarefas para um novo ciclo completo.

Sistema visual: A gestão visual em uma organização *Lean* é primordial para a equipe saber o que está em processo em determinado momento, uma prática conhecida para apresentar o *status* do trabalho é o *WIP* (*Wok-In-Progress)*. Para isso são utilizados quadros ***kanban*** que são uma representação do trabalho a ser realizado em forma de cartões. Uma outra prática muito importante é a utilização de *Heinjunka* para nivelamento da produção, por tipo e quantidade, utilizando-se como base a etapa mais lenta do processo, dessa forma a equipe tem condições de saber qual será a próxima sequência de trabalho, garante-se dessa forma a estabilidade pela menor capacidade.

Outra prática muito importante é o **arranjo físico** da equipe, em *Lean Manufaturing*, esse item é muito importante pois pode determinar mais tempo do produto em processo, como exemplo, pode-se citar o modelo de arranjo físico seguido pela indústria automobílistica. Em *Lean Software Development* não é diferente, o arranjo fisico contínua sendo importante, mas também há de se considerar as equipes virtuais, o processo de desenvolvimento deve ter um *layout* limpo, com um fluxo de trabalho com sequência lógica.

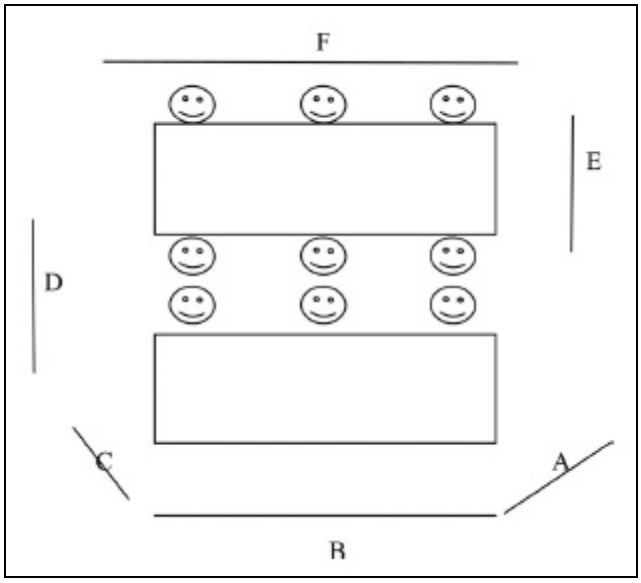

Figura 2 – O arranjo físico do escritório ((Middleton & Joyce, 2012).

Na figura 2 o arranjo físico apresenta dois quadros kanbans (A,B), quatro radiadores de informação (C,D,E,F) a disposição da equipe. Nestes são apresentados:

A = as ideias (estórias propostas pelo cliente) e seus status.
B = o progresso das estórias e tarefas.
C = se o quadro está correto, se alguém está bloqueado, o trabalho em progresso e o trabalho concluído.
D = as notificações de realese e o processo de suporte diário
E = decisões de arquitetura
F = as melhorias no sistema *Lean* e redução de complexiadade técnica ou ausência de qualidade. *O Lean Software Development* procura dessa

forma adotar um dispositivo de controle visual conhecido como Andon, que nada mais é que um alerta para os membros da equipe quando um problema ocorre.

3.1.6. Respeitar as pessoas

Respeito as pessoas é primordial para o sucesso do *Lean,* pois cria pessoas pensantes e comprometidas. Deve ser o primeiro princípio a ser considerado em uma organização que deseja praticar *Lean* (Poppendieck & Poppendieck, 2011). Se uma organização implantar todos os outros princípios colherá a sombra do *Lean,* se implantar um em particular, respeito as pessoas, posicionará as pessoas para implantarem os demais.

O *Lean* apregoa que as ideias das pessoas tem o mesmo valor, seja um engenheiro sênior ou funcionário júnior. Ainda são estimulados a trabalhar em equipe multidisciplinar, recomenda-se apenas que se tenha, ao menos, um responsável técnico ou de produto. Todo projeto em organização *Lean Software Development* tem uma liderança técnica. Em caso de implementação de processos recentes um líder de processos se faz necessário.

As pessoas são reconhecidas não pelos anos de casa, mas pelo valor agregado aos projetos, os especialistas sênior são incentivados a serem tutores dos mais novos. Em cada área do desenvolvimento deve existir um profissional com extrema competencia técnica. O êxito de um projeto é da equipe toda, nunca de um indivíduo em particular – isso inclui a avaliação de desempenho que nunca deve ser exclusivamente individual que cria competição e não cooperação entre os membros da equipe.

Não existem sistemas de ranking para promoções e aumentos de salário.

O planejamento nunca é perdido, são feitos baseados em responsabilidades e em eventos sincronizados. Os membros de equipe sabem o que é esperado do seu trabalho e os clientes também conhecem o nível de trabalho a ser entregue. Para isso o planejamento deve ser feito respeitando a capacidade de execução das pessoas.

O trabalho é autodirecionado a ponto de os membros de equipe serem capazes de saber qual a próxima tarefa a ser feita, ou em qual cada um está trabalhando, apenas com a leitura do ambiente. Nesse contexto, o ***kanban*, o *burn-down* e os *dashboards*** são importantes ferramentas de comunicação entre as pessoas.

Para finalizar, o *Lean* trata a compensação e orienta a organização a buscar uma motivação maior que o retorno financeiro para seus empregados. Entende-se que um sistema no qual as pessoas são forçadas a caminhar sentido a área gerencial apenas pelo incremento financeiro, pois se esgotaram as possibilidades na área técnica, não é sustentável, criando ranking e estimulando a competição. Quando atingem um nível salarial razoável, as pessoas procuram crescimento, um ambiente de trabalho agradável, reconhecimento, conhecimento, etc.

Resumidamente em uma organização *Lean* há evidencias claras de que a empresa está comprometida com as pessoas e as pessoas estão comprometidas com a empresa, é uma via de dois sentidos (Poppendieck & Poppendieck, 2011). Isso parte dos **contratos** que são feitos sejam da organização com o funcionário, seja da organização com cliente ou fornecedor.

3.1.7. Otimizar o todo

Uma organização guiada pelos princípios e praticas *Lean* otimiza o **mapa do fluxo de valor**, ou seja, da solicitação do cliente a entrega de uma solução, *Lean* não particiona a otimização.

O *Lean Software Development* entende a organização como um grande sistema no qual todos os sub-sistemas estão interrelacionados.

O grande equívoco neste princípio é a otimização por decomposição. A soma das partições não necessariamente representa o fluxo de valor otimizado como um todo. Como afirma Skyttner (2011), quando cada parte do sistema é melhorada independentemente, o sistema como um todo é sub-otimizado.

Um elemento importante para se entender o fluxo de valor é a elaboração de estórias completas para ter a dimensão de todo o espectro do problema. Neste sentido aplicam-se conceitos de **teoria das restrições** para implementar soluções por completo. A teoria das restrições e uma filosofia de negócio que busca alcançar o objetivo de um sistema através da melhoria da capacidade produtiva da organização, como afirma, (Katayama, 2010).

O objetivo dessa abordagem é promover a melhoria no fluxo do processo desse sistema aumentando a capacidade produtiva. Pontos de alavancagem do processo são identificados e o relacionamento entre si. Destaca-se a importancia do especialista do dominio e o responsável pelo processo na geração do mínimo de código necessário.

3.2. Considerações do Capítulo

De modo geral a análise dos princípios e práticas do *Lean Software Development* apresenta dois enfoques principais, primeiro, baseado em um conjunto de princípios e praticas e outro, baseado em dados estatísticos para se apurar o *lead time* do fluxo de valor, por exemplo.

Os demais princípios giram em torno da cultura da organização, esta define os rumos e níveis de eficiência e eficácia de seus produtos/serviços, assim como, o comportamento dos integrantes dessa organização. Por cultura, entende-se a cultura não como uma rede de comportamentos concretos e complexos, mas como um

conjunto de mecanismos que incluem controles, planos, receitas, regras e instruções que governam o comportamento (Silva & Zanelli, 2004)**.**

A organização para atender os demais princípios e práticas precisa ter uma cultura que favoreça essas práticas, a começar pelo respeito às pessoas, (Poppendieck & Poppendieck, 2011).

4. ANÁLISE DA ORGANIZAÇÃO SOB A ÓTICA DOS PRINCIPIOS *LEAN SOFTWARE DEVELOPMENT*

Para o estudo de caso são analisadas duas empresas, uma a fornecedora de serviços, a BeeIT, e na qual será avaliada o emprego dos princípios e práticas do *Lean Software Development*. A outra, a empresa cliente, a que consome os serviços ofertados, que tem apenas a sua relação com a BeeIT analisada de acordo com os princípios do *Lean Software Development*.

4.1. Caracterização da empresa analisada como fornecedora

A empresa alvo é uma consultoria em soluções de tecnologia da informação sediada em Porto Alegre, Rio Grande do Sul. Com mais de dez anos de experiência em aplicações web, inicialmente o *core business* estava voltado exclusivamente para a área da saúde e seus serviços limitavam-se à manutenção e integração de sistemas.

Após inúmeros projetos de *outsourcing* para os mais diversos segmentos a empresa passou por uma grande mudança a partir do ano de 2009, no qual seu projeto de consultoria foi bem sucedido. A partir deste momento a empresa adotou em paralelo um framework de desenvolvimento próprio. Desde então, a empresa tem concentrado esforços em pesquisas, evolução tecnológica, certificações e parcerias.

A equipe que atende à empresa denominada cliente é composta por um atendente desenvolvedor, um analista de sistemas e um coordenador.

Segundo o proprietário da empresa, a BeeIT pratica o *Scrum*, e há elementos para isso. A empresa pratica o desenvolvimento iterativo e incremental, recebe *o product backlog* e assim determina a *sprint*, a empresa também tem as reuniões para planejamento da *sprint* e discussão entre a equipe de desenvolvimento. Quanto aos papéis de cada um, não é possível afirmar se está claro para os membros da

empresa quem é o *scrum master* ou o *product owner* – apesar de a empresa afirmar que para equipe de desenvolvimento os papéis estão claros.

4.2. Caracterização do cliente

A empresa denominada cliente é uma instituição sem fins lucrativos localizada em São Paulo, capital. Com prestação de serviços de atendimento ambulatorial, medicina ocupacional e palestras para trabalhadores da construção civil.

São realizados cerca de 40 mil atendimentos ao mês entre ambulatoriais e de medicina ocupacional. Os usuários ambulatoriais são atendidos por meio de convenção coletiva, a maioria desses usuários é agendada de acordo com a disponibilidade médica**.** Para os atendimentos de medicina ocupacional a previsibilidade é menor e os agendamentos são realizados próximos à consulta, às vezes no mesmo dia. Para validar a situação das empresas associadas e usuários foi desenvolvida a aplicação *web* conhecida internamente como sistema de Gestão de Contratos, no qual são lançados contratos, validades, produtos e serviços adquiridos pelas empresas associadas**.** Esta aplicação é utilizada apenas internamente pelo departamento de cadastro.

Para o atendimento de medicina ocupacional foi desenvolvida também uma aplicação *web* conhecida internamente como sistema de MO, que faz o atendimento e emissão do PCMSO – Programa de Controle Médico de Saúde Ocupacional dos usuários que realizam exames a pedido das empresas associadas. Esta aplicação tem um pico de utilização altíssimo, sendo muito utilizada tanto pelas recepções quanto pelos médicos.

Foi desenvolvida também uma aplicação *web* conhecida internamente como Portal Empresa, o qual permite que as empresas façam periodicamente a atualização cadastral de seus funcionários. É com base nessa informação que a empresa faz sua contribuição. O pico de utilização do portal compreende os dez primeiros dias do mês, depois disso o uso dessa aplicação é eventual.

A aplicação *web* conhecida como sistema de faturamento auxilia no processo de faturamento mensal das empresas associadas. São cadastrados produtos, lotes de

produtos, empresas, regras de negócio e competências de faturamento. Pouco utilizada, se restringe a eventuais alterações nas regras de negócio.

4.3. Relação cliente-fonecedor

A Figura 3 apresenta em notação BPMN a representação do processo de abertura e tratamento dos chamados que incluem novos projetos e manutenção dos sistemas atuais. Para dar suporte aos sistemas utilizados pela empresa denominada cliente, a BeeIT qualificada como fornecedora, utiliza a ferramenta de controle de chamados (OComom). Sempre que um chamado é aberto pela equipe de tecnologia da empresa cliente, a BeeIT é notificada por email pelo OComom.

Para atender às solicitações de chamados, um esquema de atendimento, descrito na figura 3, é implementado pela empresa BeeIT. Um atendente monitora a caixa de entrada de chamados. A cada recebimento de um chamado, este avalia se o mesmo é urgente, sendo, é comunicado verbalmente e é repassado ao responsável pelo desenvolvimento. Este por sua vez, providência a alocação de um desenvolvedor.

Por outro lado, se o chamado é avaliado como não urgente, a solicitação aguarda a priorização do responsável pelo desenvolvimento (é analisada a severidade do chamado). Uma vez atendida à solicitação, o desenvolvedor devolve ao cliente para teste. Se considerada atendida a solicitação, o chamado é encerrado. Caso contrário, o chamado retorna à caixa de entrada e todo o fluxo de atendimento é refeito.

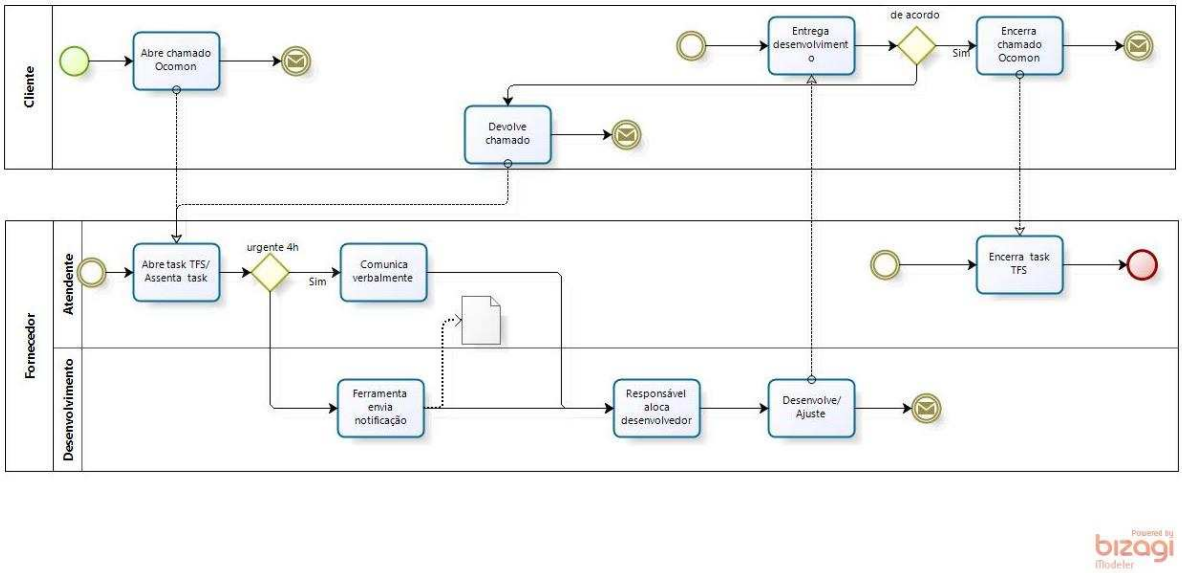

Figura 3 - Representação da relação cliente-fornecedor

4.4. Análise dos dados

Para caracterização da BeeIT em relação à utilização dos princípios e práticas *Lean Software Development foram* extraídos do OComom os chamados com status de "fechado" no período de 13/08/2010 a 14/08/2012. Procedeu-se a uma análise estatística dos dados, tendo como base o princípio entrega rápida, como sugerido pela a abordagem *Lean.*

Para os demais princípios foram feitas entrevistas e enviados questionários para entender o processo de desenvolvimento da BeeIT. Quando a resposta não era satisfatória, uma pergunta direcionada era feita na tentativa de colher a informação necessária a avaliação.

4.4.1. Eliminar desperdício

Na relação cliente-fornecedor a empresa fornecedora não tem uma equipe e processos que assegurem a qualidade de *software*, mais especificamente uma área de testes.

Sempre que uma *release* é liberada, ou um chamado devolvido, cabe a empresa cliente a realização dos testes unitários e de integração, para em seguida se fazer a homologação.

Os números apresentados no item (4.4.5 – entrega rápido) evidenciam isso, o tempo da abertura ao fechamento do chamado e o número de releases necessários para fechar o chamado.

A empresa cita que a utilização de especificação detalhada auxilia no entendimento dos requisitos, haja vista que estão homologados pelo cliente. No entanto, na relação cliente e fornecedor o tempo ajustado entre a abertura de uma solicitação e sua respectiva entrega é o mapa de fluxo de valor, mas essa informação não elimina desperdício, se da solicitação à entrega existem status que deixam os chamados em estado de hibernação.

O *Lean Software Development* pode ser empregado com diferentes propósitos segundo Wang (2011), pode ser implementado puro ou combinado a outras metodologias ágeis. A BeeIT pode aproveitar essa compatibilidade do *Lean Software Development,* para, por exemplo, praticar o TDD – *Test Driven Development* com o objetivo de elevar os padrões de qualidade esperados e percebidos pelo cliente.

E, principalmente, reduzir o número de *releases* necessárias para se entregar uma solução para o cliente e eliminar essa fonte de desperdícios para equipe.

4.4.2. Integrar a qualidade

A empresa fornecedora apresenta um esforço na adoção de ferramentas que facilitem o trabalho da equipe como a adoção *do TFS - Team Foundation Server* da Microsoft para o gerenciamento do ciclo de vida de aplicações local ou remotamente. Neste período inícial foram perceptíveis os avanços em controle de versões e a capacidade de se gerar uma nova release emergencial, caso necessário.

Por outro lado, espera-se que com a utilização de ferramentas do tipo *ALM Application Lifecycle Management*, a empresa fornecedora também tenha condição de apontar preventivamente o impacto de uma solitação de mudança ou o atendimento de um chamado.

Não há evidências da utilização de padrões de projeto, registro de erros ou programas de *housekeeping*. Como a empresa não tem um enfoque em *TDD Test Driven Development*, sua capacidade de produzir, release livre de erros, está a cada 2,34 versões, conforme apresentado (4.4.5 –Entregar rápido), A BeeIT é lenta na entrega da solução e ainda não é capaz de entregar satisfatóriamente já na primeira entrega.

Integrar a qualidade, significa fazer certo da primeira vez. No caso da Beeit, portanto, esse item não é coberto pela empresa. Apesar de a empresa destacar a utilização do *Scrum* como abodagem de desenvolvimento, com papéis muito bem definidos na *sprint* e seu tempo de duração que não pode ser superior a uma semana. Não há evidências da contribuição do *Scrum* para o princípio Integrar a qualidade e como o retrabalho é gerenciado pela equipe.

4.4.3. Criar conhecimento

Destaca-se a iniciativa da empresa fornecedora na utilização da ferramenta *Wiki* para disseminação do conhecimento na empresa. Foi adotada a *DokuWiki* para documentação em função da simplicidade, alta versatilidade, não requer uma banco de dados e é de fácil manutenção entre outros atributos.

A empresa destaca-se pela informalidade na interação com a empresa cliente. Sempre que uma dúvida ou empecilho no projeto são identificados, um contato é feito para esclarecer e solucionar o problema.

Há ainda, segundo a empresa fornecedora, como ponto forte, as reuniões frequentes, nas quais há a oportunidade para disseminação de conhecimento.

Um ponto importante nesse princípio é a interação entre o profissional sênior e um profissional júnior, segundo evidência a BeeIT, essa interação é estimulada no ambiente interno, isso está alinhado ao que sugere o *Lean Software Development.*

Esse princípio carece de maior evidência de dados, pois foram expressos neste trabalho com base em respostas do questionário enviado ao proprietário da BeeIT.

4.4.4. Adiar Comprometimentos

As especificações funcionais partem do cliente para o forncedor frequentemente com o *time-to-market* próximo da implementação. Não há por parte do cliente um planejamento quanto a introdução de novas regras de negócio, exceto as regulamentações propostas por orgãos governamentais, no qual é possível estabelecer um planejamento.

Diante desse cenário a empresa fornecedora destaca as reuniões realizadas semanalmente para planejamento da *sprint* na qual eles priorizam as estórias daquela semana. Não há menção a um plano de pivotagem quando necessário.
A empresa fornecedora em geral não tem uma margem de segurança para que as estórias amadurecam com as iterações. Isso ocorrre em função da pressão das áreas de negócio sobre a área de tecnologia, que consequentemente pressiona a empresa fornecedora para que a entrega ocorra em menor tempo. Portanto, prazo é fator limitante para o prosseguimento de boas práticas sejam elas em *Lean Software Developement* ou em qualquer outra metologia ágil.

Pode-se dizer que diante das condições impostas à empresa fornecedora, esta consegue ter a discplina para planejar a *sprint* e quais estórias serão priorizadas, no entanto, não ficou claro qual é o plano de pivotagem em caso de alguma mudança de direção durante a semana de desenvolvimento.

4.4.5. Entregar rápido

Foram classificados os chamados com status de fechado no período de 13/08/2010 a 14/08/2012. Para este trabalho os chamados foram separados por módulos e severidade, pois essa separação, a priori, não existe no Ocomom, onde os chamados são apresentados apenas por período. Apresentar os chamados por módulo e severidade poderia, instantaneamente, indicar qual aplicação apresenta comportamento anormal.

No momento em que um colaborador da TI da empresa cliente abre um chamado de solicitação para a BeeIT, uma classificação do chamado é inserida no mesmo. A

Tabela 2 traz o SLA para cada um dos sistemas (ou módulos) especificados anteriormente.

SLA por Sistema

Service Level Agreement por módulo				
	(Sev 1- Altíssima)	(Sev 2- Alta)	(Sev 3- Média)	(Sev 4- Baixa)
Faturamento - Problema	4h	1d	3d	14d
Contratos - Problema	4h	1d	3d	14d
MO – Problema	4h	1d	3d	14d
Portal Empresa - Problema	4h	1d	3d	14d

Tabela 2 – SLA por Sistema

Pode-se observar na Tabela 2, que não há uma priorização dos módulos para o cliente, ou seja todos os tempos são iguais para cada tipo de severidade independente do módulo. Isso significa que a organização fornecedora não os classifica de acordo com a maturidade de cada módulo.

Ressalta-se que essa classificação de severidade é fixa no sistema de chamados OComon para todos os fornecedores e aplicações.

A Tabela 3 traz os dados coletados no período considerado para análise.

Tabela 3 - Chamados por sistemas e severidades

Número de chamados por sistemas e severidades						
	(Sev 1-Altíssima)	(Sev 2-Alta)	(Sev 3-Média)	(Sev 4-Baixa)	Total	%
Faturamento-Prolema	1	6	8	2	17	4,96
Contratos - Problema	10	25	33	18	86	25,07
MO – Problema	13	17	64	86	180	52,48
Portal Empresa - Problema	6	6	34	14	60	17,49
Total					343	100,00

Conforme observado na Tabela 3, o módulo de MO - Problema apresenta um alto número de chamados em relação aos demais. Este módulo, além de ter frequência de utilização diária, estava passando por uma reformulação no período com a adição de novas funcionalidades, o que explica os 52,48% do total dos chamados.

Ainda de acordo com a Tabela 3, o módulo Contratos também apresenta elevado número de chamado. Este módulo tem grande variação dos requisitos em função da alternância e adequação das regras de negócio, o que é a possível razão dos 25,07% do total de chamados.

O módulo Portal é responsável por 17,49% do total de chamados (Tabela 3). Esse módulo foi desenvolvido há pouco tempo e está gradativamente tendo o incremento de novas funcionalidades.

Para cálculo do *lead time* por módulo foi considerado o tempo de abertura do chamado até o seu fechamento, independentemente da severidade (Sev 1, Sev 2, Sev 3. Sev 4), como determinado pelo *Lean*.

Na relação cliente-fornecedor dependendo do status do chamado, o SLA para de contar para uma das partes, por exemplo, se um chamado aberto pelo cliente tem o status alterado pela empresa fornecedora para “aguardando mais informação”, esse tempo não é contabilizado para a empresa fornecedora. No cômputo final, apenas o período em que o chamado ficou com status de “em atendimento” terá o tempo contabilizado.

Portanto, sob a perspectiva do fornecedor, ele somente terá a contagem do tempo contra ele, caso o status que tenha sido atribuído permitir essa aferição. Na (Tabela 4) se o status tem a contagem do tempo igual a “S” então a contagem do tempo está sendo realizada.

Nas versões superiores do Ocomom é possível determinar quanto tempo o chamado ficou sob a responsabilidade da empresa cliente ou fornecedora.

Tabela 4 - Contagem do tempo por status

Contagem do tempo por status	
Status	Conta o tempo
Agendado com usuário	N
Agendado com o usuário na abertura	N
Aguardando atendimento	S
Aguardando especificação dos requisitos pelo usuário	N
Aguardando estimativa de custo/prazo	N
Aguardando *feedback* do usuário	N
Aguardando mais informações	N
Aguardando priorização	N
Aguardando retorno do fornecedor	N
Cancelado	N
Com backup	N/A
EF aguardando aprovação pelo usuário	N
Em análise de viabilidade	N
Em aprovação pelo usuário	N
Em aprovação de custo/prazo	N
Em atendimento	S
Em desenvolvimento	S
Em especificação funcional	N
Em estudo	N
Em homologação	N
Em implantação	N
Em seleção de fornecedor	N
Em testes	N
Encaminhado para operador	S
Entregue para testes	N
Interrompido para atender outro chamado	N/A
Prontuário encaminhado para setor	N/A
Suspenso	N/A

N= Não conta o tempo S= Conta o tempo N/A = Não se aplica

Tabela 5 – Tempo médio da abertura ao fechamento/dias

Média de tempo da abertura ao fechamento/dias				
	(Sev 1- Altíssima)	(Sev 2- Alta)	(Sev 3- Média)	(Sev 4- Baixa)
Faturamento - Problema	138	152	188	3
Contratos - Problema	110	52	45	14
MO – Problema	37	136	30	10
Portal Empresa - Problema	1	4	5	7

Na Tabela 5 apresenta-se a média de tempo da abertura ao fechamento dos chamados dividido pelo número de chamados.

Conforme mostrado na Tabela 5, pode-se aferir que a organização fornecedora respondeu satisfatoriamente (em verde e amarelo) aos chamados de severidade baixa para todos os módulos, ou seja, abaixo do *SLA* proposto, apenas o módulo de contratos apresentou o mesmo número, entre dias corridos da abertura ao fechamento e *SLA.*

Em todos os outros módulos, independentemente da severidade, o tempo médio da abertura até o fechamento do chamado foi muito superior (em vermelho) ao estabelecido em *SLA.*

Pode-se concluir que a empresa fornecedora não entrega rápido as solicitações, com severidade altíssima, alta e média que a ela são feitas pela empresa cliente, o que é gravísimo; pois o *time to market* não é atendido. Dessa forma, está em desacordo com o que apregoa a abordagem *Lean Software Development* que determina que o tempo decorrido entre a solicitação de um pedido e sua respectiva entrega seja o menor possível.

O Ocomon, a ferramenta utilizada para gestão dos chamados não permite a rápida visualização da prioridade do chamado de acordo com SLA definido entre fornecedor e cliente; e o tempo de atendimento da solicitação é ajustado de acordo

com o status que determina a contagem ou não do tempo. Esse ajuste que a ferramenta faz no tempo do chamado, da abertura à entrega, dificulta completamente a visualização do tempo real da abertura à entrega.

Portanto, a demora para entrega de uma solicitação pode estar tanto no processo de desenvolvimento da BeeIT como no sistema de controle de chamados, Ocomom, que não apresenta a informação priorizada e dificulta o gerenciamento dos chamados.

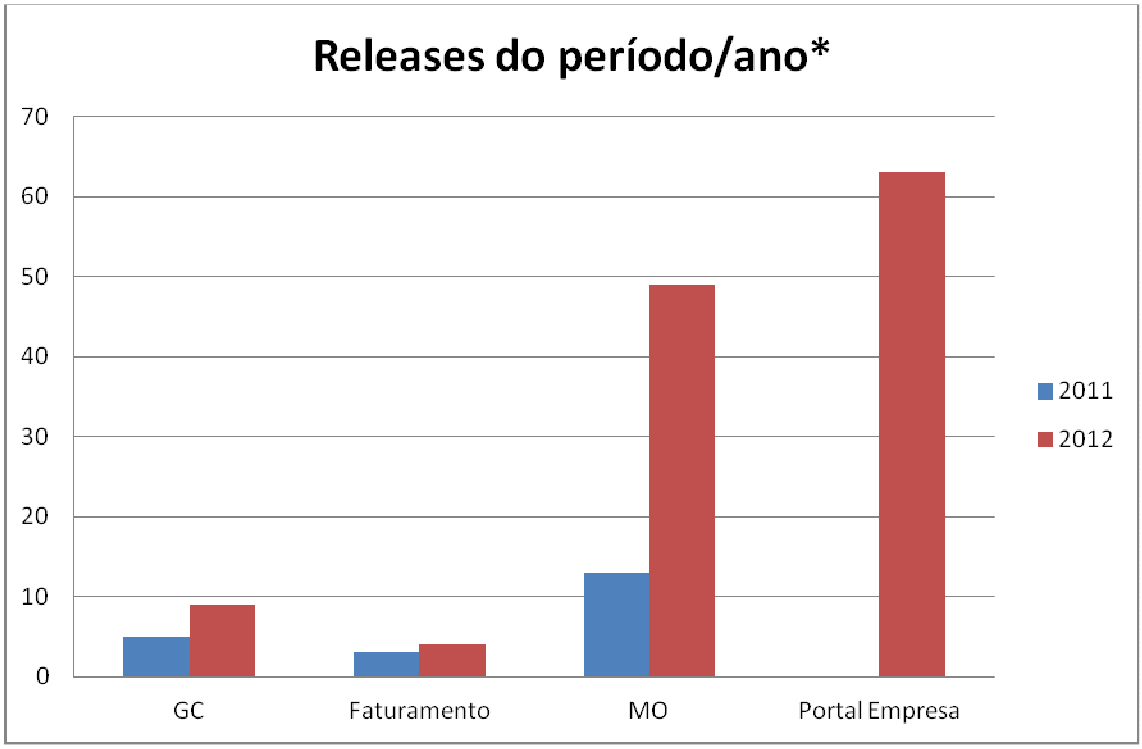

Figura 4 – Realeses do período/ano

Tabela 6 – Índice de realeses do período.

Releases do período*		
	Total	Índice
Faturamento - Problema	7	2,43
Contratos - Problema	14	6,07
MO – Problema	62	2,90
Portal Empresa - Problema	63	0,95
Total	146	2,34

*aproximado

No gráfico apresentado na Figura 4 e conforme observado na Tabela 6 foi levantado o número de releases aproximados para cada módulo. Importante destacar que a empresa fornecedora no início de suas operações junto ao cliente não tinha um repositório específico para liberação das releases. Adicionalmente, alguma versão pode ter sido entregue de forma alternativa ao repositório, por email, por exemplo.

Nota-se que de forma geral, considerando todos os chamados e o tempo entre a abertura e o fechamento, a empresa fornecedora precisa de 2,34 releases para atender aos chamados.

Destaca-se o número elevado de releases para o módulo de gestão de contratos, 6,07 releases para entregar valor ao cliente. Por outro lado, o módulo Portal Empresa apresenta um número próximo de 1, isso significa que a cada versão a empresa consegue entregar os chamados abertos. No entanto, por se tratar de um módulo novo, conclui-se que com a evolução dos testes novos bugs foram encontrados, uma vez que a especificação funcional estava pronta e o módulo deveria ser entregue acabado, sem iterações. Houve também uma evolução de releases de 2011 para 2012 além de uma nova aplicação, Portal da empresa.

4.4.6. Respeitar as pessoas

Em 01/12/2011 foi estabelecido o contrato que delimita a parceria entre a empresa fornecedora de serviços de manutenção e suporte de sistemas, a Beeit, e o cliente.

O contrato prevê a consultoria e suporte técnico de todos os sistemas *Web*. Para estes serviços existe um valor fixo mensal, não existem horas mínimas ou máximas de uso do cliente, tanto para o serviço da consultoria, treinamento a distância, como manutenção dos *softwares*.

O contrato também prevê valores de horas pré-fixadas para novos projetos. São previstas valores para programadores, analistas de sistemas e analista de negócios.

O novos projetos consistem na implementação de novas funcionalidades do sistema, ou até mesmo novos sistemas. Para isso é encaminhado pelo cliente um documento com requisitos e funcionalidades, denominado EF – Especificação Funcional e, a partir deste momento, o fornecedor devolve a estimativa de horas necessárias para o desenvolvimento.

O valor total de cada novo projeto tem como componente o número de horas estimado e acordado para o desenvolvimento, multiplicado pelos valores estipulados no contrato para cada tipo de recurso empregado no projeto.
No momento da elaboração do contrato não houve a inclusão de uma clausula que assegurasse o rejuste períodico, anual ou por incremento de novos sistemas ao contrato de suporte e manutenção dos sistemas.

Existia a expectativa inicial de que no futuro os sistemas fossem compartilhados ou comercializados com outras empresas do mesmo segmento e com isso novos contratos de suporte e manutenção dos sistemas. Destaca-se que mesmo o cliente, atualmente, não compartilha os seus sistemas com outras empresas.

Em linhas gerais esta é a parceria estabelecida entre cliente-fornecedor para manutenção e suporte aos sistemas *Web*.

Um aspecto citado pela empresa que vai ao encontro do filosofia *Lean Software Development* é o respeito a opnião ou sugestão de qualquer membro da equipe independentemente da posição hierarquica da pessoa, como afirma, (Poppendieck & Poppendieck, 2011).

Na ausência de maiores evidências para esse princípio, esta análise concentra-se na relação cliente-fornecedor, ainda como afirma, (Poppendieck & Poppendieck, 2011)**,** Se contratos são elaborados considerando valores fixos, perde-se a excência do *Lean Software Development* que é respesito as pessoas. A BeeIT, deve rever este contrato e atentar-se na elaboração de novos para que valores fixos não sejam praticados.

4.4.7. Otimizar o todo

Como demonstrado na representação da relação cliente-fornecedor (Figura 3) após o envio da Especificação Funcional para a empresa fornecedora nenhum documento é recebido de volta com a análise técnica do impacto no código.

Essa mitigação de impacto fica muito dependente do analista de negócio da empresa cliente na hora da elaboração da Espefificação Funcional. As dúvidas de projeto são sanadas durante a fase de desenvolvimento e essas alterações muitas vezes não ficam registradas para o cliente.

Compreende-se também como afirmardo no 4.4.4 – Adiar Comprometimentos que a BeeIT muitas das vezes não tem margem para um planejamento eficaz, muito menos para uma análise do que foi proposto em Especifição Funcional; cabendo apenas o desenvolvimento em si.

A forma de contrato também impede que a empresa fornecedora dispense horas analisando o impacto da solicitação e consequentemente otimizando o todo, o preço é fechado.

Nota-se neste princípio o impacto de princípios anteriores, a BeeIT trabalha de forma reativa as solicitações do cliente.
Como descrito neste trabalho a empresa cliente exerce uma pressão muito forte para que o trabalho seja entregue sem a contrapartida contendo a análise técnica da empresa fornecedora. Não existe uma margem de segurança para um planejamento eficaz.

4.5. Considerações do Capítulo

Este capítulo apresentou os resultados da experimentação feita, que envolveu a análise norteada pelos princípios do *Lean Software Development*. Os questionários empregados na análise são apresentados nos Anexo A, B e C.

5. CONSIDERAÇÕES FINAIS

Durante o desenvolvimento deste trabalho foi identificado que o *Lean Software Development* tem um range muito grande de implementação com diferentes propósitos, a identificação desses princípios depende, fundamentalmente, do contexto da organização, conforme aponta ((Middleton & Joyce, 2012), 2012).

É de extrema importância identificar os objetivos que norteiam a estratégia da empresa, por exemplo, redução de custos, entregar mais rápido ou disseminar o *Lean* perante aos demais departamentos da empresa. Observa-se que a prática de outras metologias ágeis não assegura a implementação do *Lean Software Development* mas o contrário é verdadeiro, como constatado neste estudo de caso.

Neste trabalho, por meio das práticas em metodologias ágeis da BeeIT, mais especificamente o *Scrum*, foi feita a análise para identificar os sete princípios do *Lean Software Development*. O princípio entregar rápido foi o que apresentou maior facilidade de análise haja vista que os dados estavam disponíveis e somente foram tabulados. Para os demais princípios foi necessário estimular o entrevistado para que a informação desejada fosse apresentada. Em virtude da distância entre pesquisador e empresa analisada, o contexto da empresa e os demais princípios carecem de maior profundidade.

Diante disso, conclui-se que a BeeIT não consegue visualizar a sua principal fonte de retrabalho e consequentemente **Eliminar desperdício**. A empresa cita que as Especificações Funcionais auxilíam no entendimento dos requisitos, no entanto, na prática, não há evidência que as EF’s aumentam o entendimento dos requisitos. A empresa não tem um plano de testes para as suas principais aplicações. No princípio **Integrar a qualidade** a empresa também não pratica a abordaem *Lean*

Software Development a medida que não faz certo da primeira vez e são necessárias 2,34 releases para atender aos chamados. No princípio **Criar conhecimento** a BeeIT apresenta aspectos positivos em relação a abordagem, informalidade no contato com cliente. A a adoção de ferramenta *Wiki* para a disseminação de conhecimento entre os membros da equipe é um facilitador nos projetos, mas o *Lean* sugere muito mais que isso, a proposta é que os membros da equipe tenham tempo para criar esse conhecimento e a preocupação da equipe não seja apenas entregar rápido para satisfazer o cliente. **Adiar comprometimentos**, este princípio simplesmente inexiste para o fornecedor, via de regra, as EF's chegam para o desenvolvimento já pressionadas pelo *time to market,* não existe um plano de pivotagem caso tenha alguma mudança a ser realizada. **Entregar rápido**, não apresenta uma cadência e velocidade do pedido a entrega da solicitação. **Respeitar as pessoas**, a interação entre um profissional sênior e um júnior nos projetos é ponto de destaque nas reuniões frequentes, na ausência de mais dados para evidenciar esse princípio, este estudo atenta-se para a relação cliente-fornecedor, pois o *Lean Software Development* sugere que os contratos tem que levar em consideração as pessoas, conforme descreve (Poppendieck & Poppendieck, 2011), Se contratos são elaborados considerando valores fixos, perde-se a excência do *Lean Software Development* que é respesito as pessoas. A BeeIT não consegue práticar essa abordagem em função do seu principal cliente impor um contrato favorável a ela e tão desigual para a fornecedora, ainda conforme (Poppendieck & Poppendieck, 2011), a essência do *Lean Software Development* é o respeito as pessoas, os autores sugerem que a implementação da abordagem deve começar pelo respeito as pessoas os demais princípios são consequência. **Otimizar o todo**, esse princípio também inexiste para a empresa fornecedora, muito em função do não atendimento a outros princípios, a empresa trabalha sob pressão de prazos e custos, dessa forma é improvável que consiga entregar, por exemplo, um documento técnico com análise de impacto no código ou apresente sugestões de melhoria ao cliente, a única preocupação, como apresentado anteriormente, é entregar o que foi solicitado apenas.

5.1. Contribuições do Trabalho

O trabalho apresenta uma definição concisa e clara da abordagem *Lean Software Development* dentro do conceito de metodologias ágeis distinguindo-a das demais.

No exemplo apresentado como experimento foi possível compreender os diferentes propósitos de implantação da abordagem e que, o *Lean Software Development* depende fundamentalmente da alta administração, principalmente, no que diz respeito aos princípios e que apenas a adoção de práticas no dia a dia não configura a implantação da abordagem.

5.2. Trabalhos Futuros

O estudo realizado procurou identificar na empresa analisada praticante de métodos ágeis, elementos dos princípios e práticas *Lean Software Development*. Propõe-se que outros trabalhos nessa linha sejam desenvolvidos, pois no experimento aqui discutido houve várias limitações que prejudicaram um pouco a obtenção de um resultado mais abrangente. Para futuros experimentos sugere-se que o pesquisador tenha um contato mais próximo com a empresa, conhecendo-a no dia-a-dia, além de ter o relato de várias pessoas dentro da empresa, preferencialmente com diferentes papéis. Assim, será possível uma melhor avaliação do comprometimento das pessoas que executam o serviço e da alta administração, que fundamental no estabelecimento dos princícpios *Lean Software Development*. O questionário utilizado no experimento apresentado nesta monografia pode auxiliar bastante nesta nova empreitada.

Sugere-se que o estudo proposto considere a implementação dos princípios em empresas novatas que desenvolvem produtos em ambientes de extrema incerteza, segundo a definição de (Ries, 2012) para *Lean Startups.*

Também seria apropriado um trabalho que investigasse a aplicação do *Lean Software Development* em organizações focadas em manutenção de *software.*

REFERÊNCIAS BIBLIOGRÁFICAS

Agile Alliance. (17 de 08 de 2012). *agilealliance.org*. Acesso em 01 de 03 de 2013, disponível em AgileAlliance: cwww.agilealliance.org/the-alliance/the-agile-manifesto/the-twelve-principles-of-agile-software/

Beck, K. (2003). *Test Driven Development: By exemple.* Boston: Pearson Education .

Business Model Generation. (15 de 11 de 2012). *Business Model Generation*. Acesso em 15 de 11 de 2012, disponível em Business Model Generation: http://www.businessmodelgeneration.com/

David Raffo, M. M. (2010). Integrating Lean Principles with Value Based Software Engineering.

Demarco, T. (2001). *Slack: Getting past burnout, busywork, and muth of total efficiency.* New York: Broadway books.

Katayama, E. T. (2010). *A contribuição da indústria da manufatura no desenvolvimento de software.* São Paulo.

Middleton, P., & Joyce, D. (2012). Lean Software Development: BBC Worldwide Case Study. *IEEE* .

PMI. (2004). *Guia PMBoK.* Pensilvania: PMI.

Poppendieck, M., & Poppendieck, T. (2011). *Implementando o Desenvolvimento Lean de Software.* Porto Alegre: Bookman.

Pressman, R. S. (2009). *Engenharia de Software - Uma abordagem profissional.* Porto Alegre: Atlas.

Ries, E. (2012). *A Startup enxuta.* São Paulo: Lua de Papel.

Silva, N., & Zanelli, J. C. (2004). *Psicologia, organizações e trabalho no Brasil.* Porto Alegre: Artmed .

Skyttner, L. (2001). *General systems theory: ideas e aplications.* New Jersey: World Scientific Publishing Company .

Swaminathan, B., & Jain, K. (2012). Implementing the Lean concepts of Continuos Improvement an Flow on an Agile Software Development Project - An Industrial Case Study. *Agile India* .

Wang, X. (2011). The Combination of Agile and Lean in Software Development: An Experience Report Analysis. *Agile Conference* .

APÊNDICE A – Questionário os processos da BeeIT

1. Quando um chamado (tipo problema) é aberto para Beeit, como ele é tratado internamente, como vocês se organizam?

2. Tem responsável (eis) pela avaliação dos chamados?

3. Qual é o fluxo para cada um dos módulos, Gestão de contratos, MO, Portal da Empresa e Faturamento, ou é o mesmo? Estamos considerando apenas chamados do tipo problema, para cada um dos módulos.

4. Considerando a severidade (Sev 1-Altissima, (Sev 2-Alta) (Sev 3-Média) e (Sev 4-Baixa), o fluxo é o mesmo?

5. Em casos de impedimentos, “dificuldades”, do negócio, como vocês se comportam internamente, tem um processo para estes casos?

6. Como é o processo de documentação, primeiro codifica depois documenta, ou vocês o fazem em paralelo? Utilizam alguma ferramenta?

7. Um colega de trabalho sabe no que o outro está trabalhando no momento?

8. Cada integrante da equipe desempenha apenas a sua função, ou na medida do possível, todo mundo faz um pouco de tudo? Existe alguém específico para uma função.

9. Quais fatores são analisados para a liberação de uma release?

10. Vocês tem (ou praticam) algum modelo de desenvolvimento, qual?
11. Tem alguma informação que você gostaria de adicionar e que não foi perguntado sobre o processo?

APÊNDICE B – Questionário para avaliar o contrato de serviço

1.Quando o contrato entre a BeeIT e o cliente foi celebrado?

2.E qual é o formato em linhas gerais?

3.Quando celebrado, o que estava previsto? Teve alteração ao longo do tempo?

4.Já aconteceu de a Beeit ter que cumprir o contrato, mas ter prejuízo financeiro porque os requisitos mudaram e o valor do contrato não foi renegociado?

5.Quer acrescentar algo que não foi perguntado?

APÊNDICE C – Questionário os princípios *Lean Software Development*

1.Eliminar desperdício: Como vocês identificam o desperdício e o eliminam? Vocês discutem e implementam novas formas de se fazer o trabalho?

2.Integrar a qualidade: Quais são os processos, métodos, boas praticas, testes, padrões que vocês utilizam nos projetos? Quanto mais material melhor.

3.Criar conhecimento: Aqui vou explorar a nossa forma de trabalho iterativa e incremental, os acessos remoto.

4.Adiar Comprometimentos: Os requisitos do cliente são voláteis? Em geral o cliente sabe o que quer? Como vocês se 'previnem' contra a variação de requisitos?

5.Entregar rápido: este eu já fiz...mas se quiser complementar com algo é bem-vindo qualquer informação.

6.Respeitar as pessoas: descreva algum aspecto relacionado a gestão de pessoal.

7.Otimizar o todo: na sua opinião, o que o cliente espera da beeit: preço, rapidez, qualidade? Nos novos projetos vocês se envolvem na otimização do todo e não da parte?